Computer Science 3650
Take Home Exam 1
Due: Start of Class Friday March 6

Turn in one copy per team and make sure ALL names (last name first) are on ALL pages.

Ethics Policy: The only resources to be consulted in working these questions are the textbook, your notes, the instructor, and your teammates.

1. Prove that $f(n) = 4n^2+2n+3 = O(n^2)$ (10 points).

2. Suppose we are executing Merge-Sort on the following six values --- 33, 38, 42, 44, 45, 50. Give a permutation of these values such that when Merge-Sort is applied to the permutation, no values will change position until the final Merge of the left half (3 values) and the right half (3 values), and when that merge is applied, all values will change position (5 points).

3. We saw that the best case running time of insertion sort is only $\Theta(n)$, which is significantly better than the worst case running time of $\Theta(n^2)$. Does the best case running time of merge sort show a similar speedup from $\Theta(nlg\ n)$ to $\Theta(n)$ or to $\Theta(lg\ n)$? You must justify your answer (5 points)

4. When counting the number of arithmetic operations in evaluating an expression, we often make the simplifying assumption that each operation has the same cost. For example, consider the following statement from the Merge-Sort algorithm for dividing the problem into two equal problems.

   $q = \lfloor (p + r)/2 \rfloor$

   we would say that the evaluation requies a total of 3 arithmetic operations (+, / and "floor"). That is, the time cost is 3. If that statement was embedded in a loop that executed 5 times, then we would say that the statement has a time cost of 5*3 = 15 arithmetic operations.

   Now consider this equation that was used in calculating the execution time for insertion sort.

   $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$

   What is the difference in the time cost $T(n)$ in terms of the number of arithmetic operations required to compute this formula depending on whether the algorithm is expressed based on the summation approach on the left-hand side of the equation or based on the closed formula approach on the right hand side of the equation as a function of the problem size *n*? You should express your analysis using the theta (Θ) notation. (10 points)

5. Let T(n) represent the total execution time of a loop for which the loop test is true *n* times (i.e., the loop body will execute *n* times). For each possible value of T(n) listed below, give a <u>tight</u> upper bound for the time cost (in terms of O(…)) of <u>one</u> execution of the loop body

(denote this by L(n)) that makes the statement true. Answer each part separately. Each answer should be of the form L(n) = O(…). You fill in the … (5 points each, 15 points total).

a. $T(n) = O(n^2)$
b. $T(n) = O(n)$
c. $T(n) = O(n \lg n)$

6. Use the master method to give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible, and justify your answers by demonstrating how the master method applies (5 points each, 15 points total).

   a. $T(n) = 16T(n / 4) + n^2$.
   b. $T(n) = 7T(n / 2) + n^2$.
   c. $T(n) = 2T(n / 2) + n^4$.

7. The following represents pseudocode for the function *maxVal(A,first,last)* that returns the maximum value within A[first..last]. Assuming in general that A is an array with *n* elements, then the function call *maxVal(A,1,n)* would return the maximum value in the entire array.

```
maxVal(A,first,last)
    best = A[first]
    for i = first+1 to last
        if A[i] > best
            best = A[i]
    return best
```

   and of course the time cost T(n) for the function is Θ(n).

   Write a divide and conquer version of *maxVal*. In writing your solution you may assume the existence of a function *max(X,Y)* that returns the maximum of the two numbers *X* and *Y* in time Θ(1). In addition, specify the recurrence equation for T(n) for *maxVal*, and solve the recurrence equation to provide asymptotic upper and lower bounds for T(n). Are the bounds the same as the iterative version? (20 points)

8. Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in $\Theta(n^2)$ time (10 points).

9. In class we discussed an approach that yields a Θ(n) algorithm for solving the maximum subarray problem. This approach incrementally calculates the maximum subarray for A[1..j] that uses A[j], and then uses that value to calculate the maximum subarray for A[1..j+1] that uses A[j+1]. The answer is the maximum of all those *n* calculated values. Write pseudocode for a version of this algorithm that instead works from the right side of the array to the left. That is, it calculates the maximum subarray for A[j..n] that uses A[j], and then use that result to calculate the maximum subarray for A[j-1..n] that uses A[j-1] (10 points).